Predicting heart disease using machine learning

This notebook looks into using various python-based machine learning and data science libraries in an attempt to build a machine learning model capable of predicting whether or not soemone has heart disease based on their medical attributes.

we're going to take the following approach:

1. problem definition
2. data
3. Evaluation
4. features
5. experimentation

1. Problem Definition

In a statement,

> Given clinical parameters about a patient, can we predict whether or not they have heart disease?

2. Data

originally from kaggle

3. Evaluation

> Ifwe can reach 95% accuracy at predicting whether or not a patient has heart disease during the proof of concept, we'll pursue the project

4. Features

This is where you'll get different information about each of the features in your data.

create data dictionary

1. age - age in years
2. sex - (1 = male; 0 = female)
3. cp - chest pain type *0: Typical angina: chest pain related decrease blood supply to the heart *1: Atypical angina: chest pain not related to heart *2: Non-anginal pain: typically esophageal spasms (non heart related) *3: Asymptomatic: chest pain not showing signs of disease
4. trestbps - resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically cause for concern
5. chol - serum cholestoral in mg/dl serum = LDL + HDL + .2 * triglycerides above 200 is cause for concern fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false) '>126' mg/dL signals diabetes restecg - resting electrocardiographic results 0: Nothing to note 1: ST-T Wave abnormality can range from mild symptoms to severe problems signals non-normal heart beat 2: Possible or definite left ventricular hypertrophy Enlarged heart's main pumping chamber thalach - maximum heart rate achieved exang - exercise induced angina (1 = yes; 0 = no) oldpeak - ST depression induced by exercise relative to rest looks at stress of heart during excercise unhealthy heart will stress more slope - the slope of the peak exercise ST segment 0: Upsloping: better heart rate with excercise (uncommon) 1: Flatsloping: minimal change (typical healthy heart) 2: Downslopins: signs of unhealthy heart ca - number of major vessels (0-3) colored by flourosopy colored vessel means the doctor can see the blood passing through the more blood movement the better (no clots) thal - thalium stress result 1,3: normal 6: fixed defect: used to be defect but ok now 7: reversable defect: no proper blood movement when excercising target - have disease or not (1=yes, 0=no) (= the predicted attribute) Note: No personal identifiable information (PPI) can be found in the dataset.

   It's a good idea to save these to a Python dictionary or in an external file, so we can look at them later without coming back here.

preparing the data

we're going to use pandas, matplotlib, and numpy for data analysis and manipulation

In [1]:
```python
# import all the tools we need
# Regular EDA(exploratory data analysis) and plotting libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# we want our plots to appear inside the notebook
%matplotlib inline

# models from scikit-learn
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# model Evaluation
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import plot_roc_curve
```

In [2]:
```python
import os
cwd = os.getcwd()
print(cwd)
```

```
C:\Users\USER\Desktop\heart-disease-project
```

Load data

In [3]:
```
df = pd.read_csv("7.1 heart-disease.csv")
df.head()
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [4]:
```
df.shape # (rows, columns)
```

Out[4]:
```
(303, 14)
```

Data Exploration (exploratory data analysis or EDA)

The goal here is ti find out more about the data and become a subject matter export on the dataset you're working with.

1. What question(s) are you trying to solve?
2. what kind of data do we have and how do we treat different types?
3. What's missing from the data and how do you deal with it?
4. where are the outliers and why should you care about them?
5. how can you add, change or remove features to get more out of your data?

In [5]:
```
df.head()
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [6]:
```
df.tail()
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

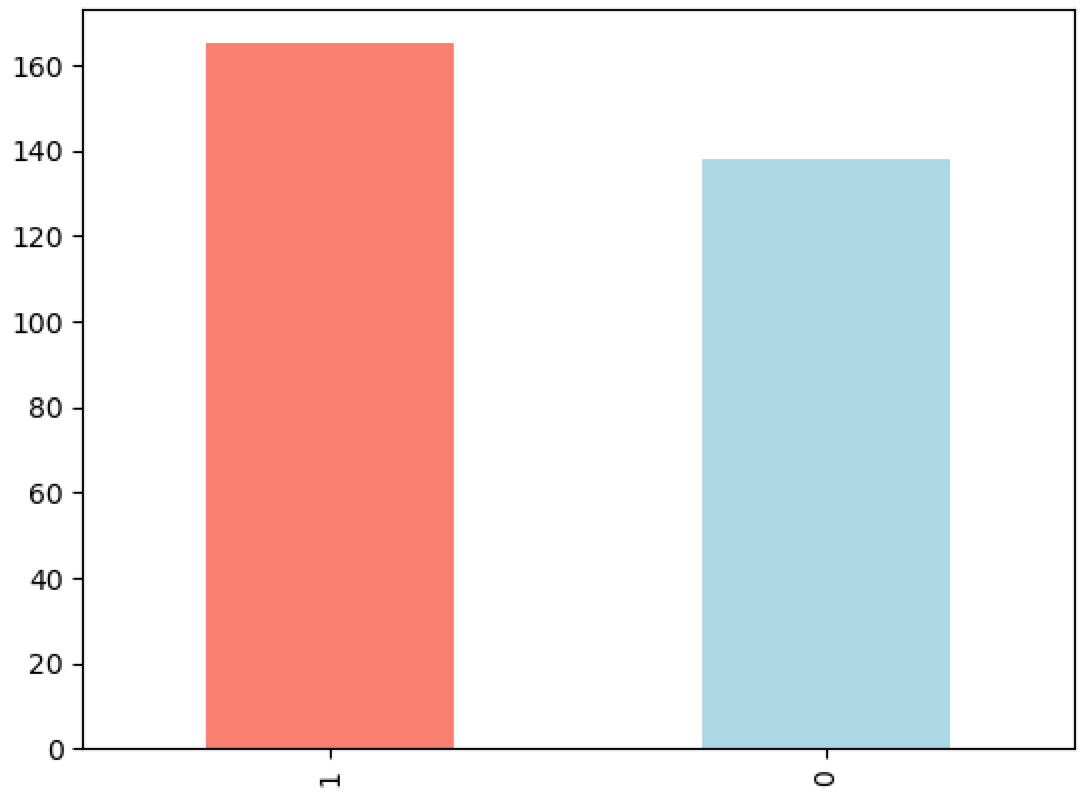
In [7]:
```
df.target
```

Out[7]:
```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

In [8]:
```
# let's find out how many of each class there
df['target'].value_counts()
```

Out[8]:
```
1    165
0    138
Name: target, dtype: int64
```

In [9]:
```
df['target'].value_counts().plot(kind = 'bar', color =['salmon', 'lightblue']);
```



In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [11]:
```
# Are there any missing values?
df.isna().sum()
```

Out[11]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [12]:
```
df.describe()
```

Out[12]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	3(
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	

Heart Disease frequency according to sex

In [13]:
```
df.sex.value_counts()
```

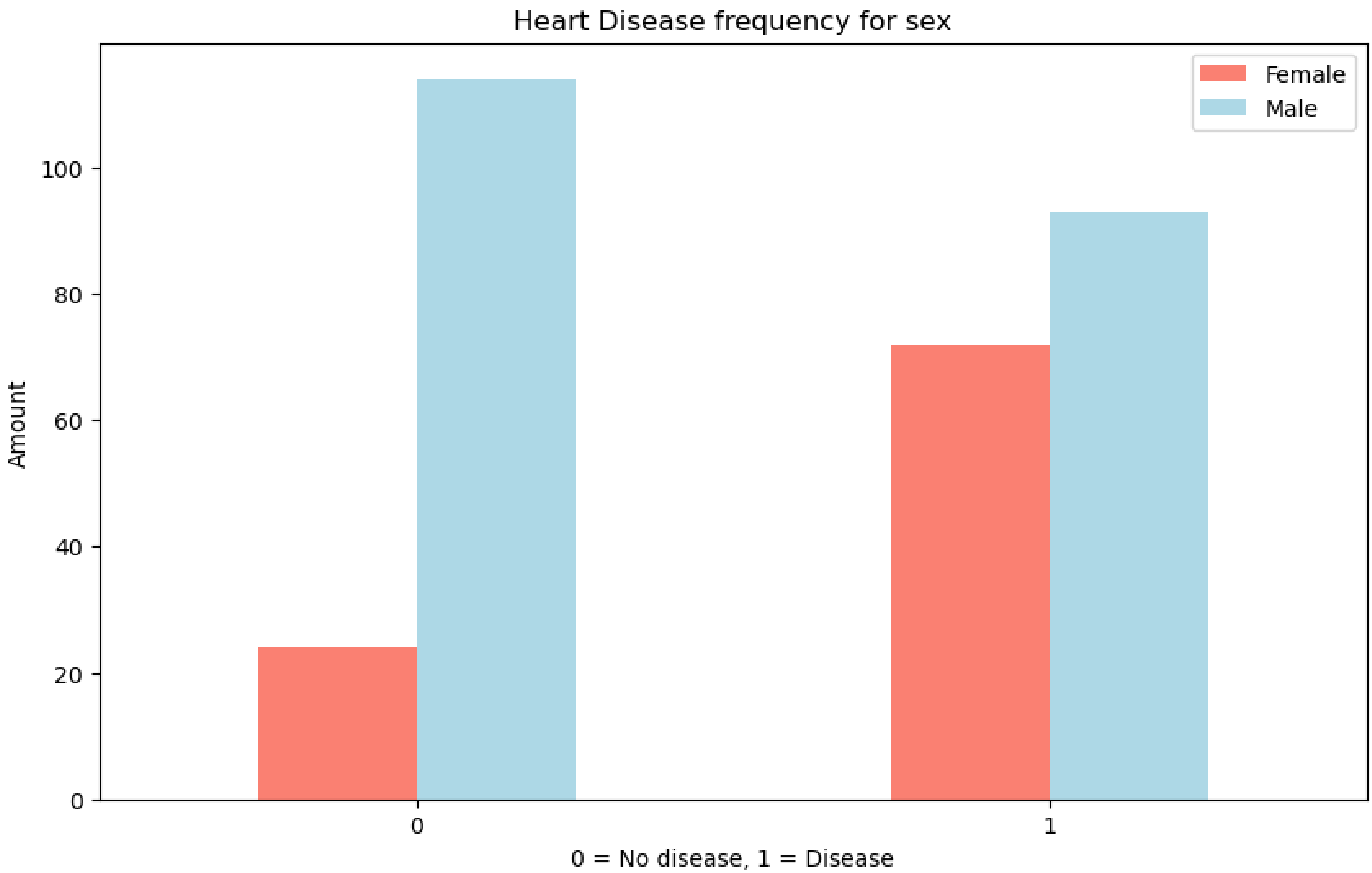
Out[13]:
```
1    207
0     96
Name: sex, dtype: int64
```

In [14]:
```
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

Out[14]:

sex	0	1
target		
0	24	114
1	72	93

In [15]:
```
# Create a plot of crosstab
pd.crosstab(df.target, df.sex).plot(kind = 'bar',
                                    figsize = (10, 6),
                                    color = ['salmon', 'lightblue'])
plt.title('Heart Disease frequency for sex')
plt.xlabel('0 = No disease, 1 = Disease')
plt.ylabel('Amount')
plt.legend(['Female', 'Male'])
plt.xticks(rotation = 0);
```



In [16]:

```
df['thalach'].value_counts()
```

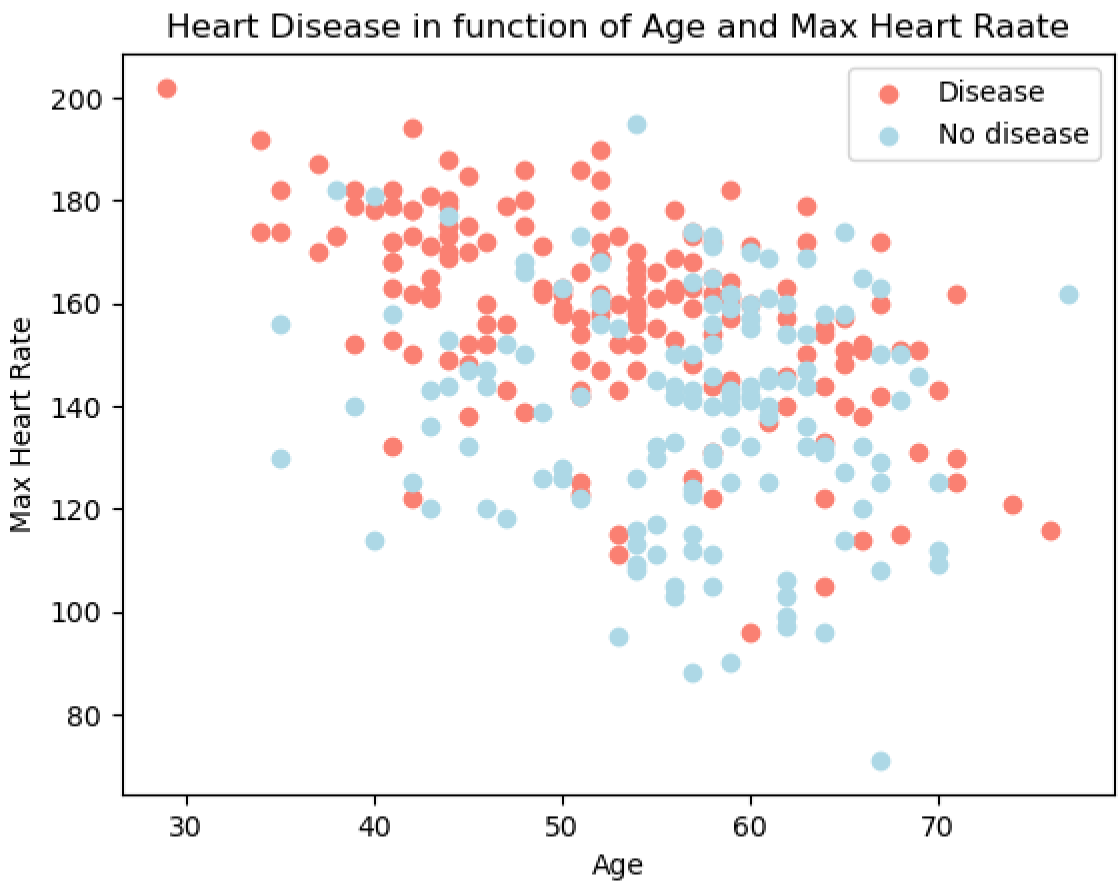
Out[16]:

```
162    11
160     9
163     9
152     8
173     8
       ..
202     1
184     1
121     1
192     1
90      1
Name: thalach, Length: 91, dtype: int64
```

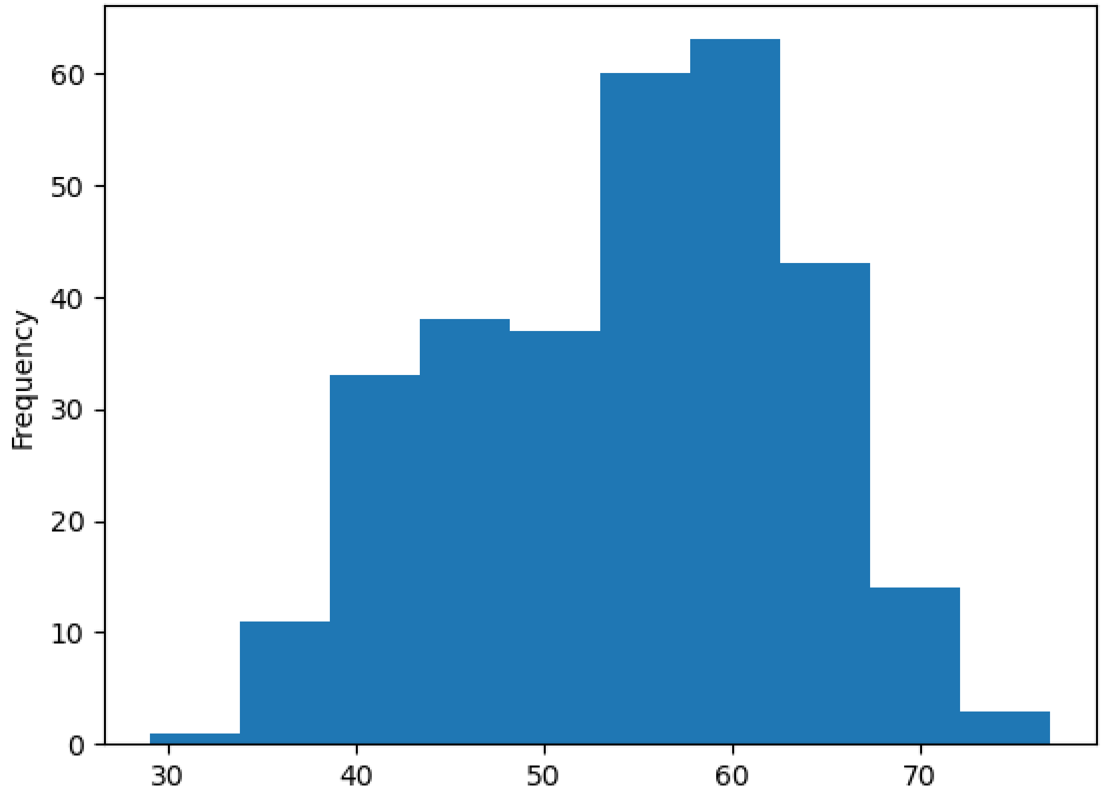
Age vs. Max Heart Rate for Heart Disease

In [17]:

```
# Create another figure
plt.figsize = (10, 6)

# scatter with positive examples
plt.scatter(df.age[df.target == 1],
            df.thalach[df.target == 1],
            c = 'salmon', label ="Disease")

# scatter with negative examples
plt.scatter(df.age[df.target == 0],
            df.thalach[df.target == 0],
            c = 'lightblue', label = 'No disease');

# Add some helpful information
plt.title('Heart Disease in function of Age and Max Heart Raate')
plt.xlabel('Age')
plt.ylabel('Max Heart Rate')
plt.legend();
```


Heart Disease in function of Age and Max Heart Raate
Disease
No disease
Max Heart Rate
Age
200
180
160
140
120
100
80
30
40
50
60
70

In [18]:
```
# check the distribution of the age column with a histogram
df.age.plot.hist();
```



Heart Disease Frequency Chest pain Type

cp - chest pain type

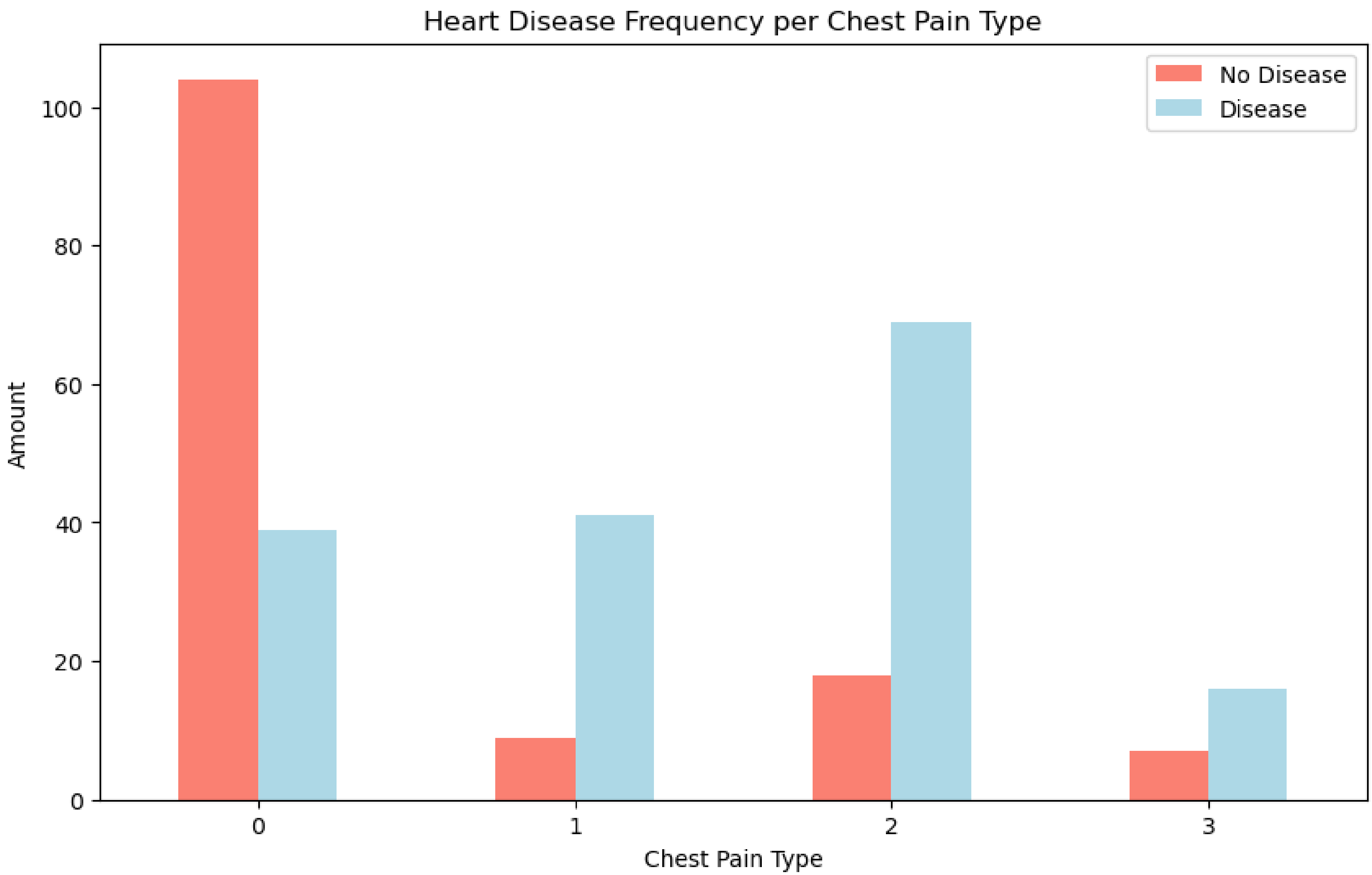
- 0: Typical angina: chest pain related decrease blood supply to the heart
- 1: Atypical angina: chest pain not related to heart
- 2: Non-anginal pain: typically esophageal spasms (non heart related)
- 3: Asymptomatic: chest pain not showing signs of disease

In [19]:
```
pd.crosstab(df.cp, df.target)
```

Out[19]:

target	0	1
cp		
0	104	39
1	9	41
2	18	69
3	7	16

In [20]:
```
# Make the crossbar more visuaal

pd.crosstab(df.cp, df.target).plot(kind = 'bar',
                                  figsize = (10, 6),
                                  color = ['salmon', 'lightblue'])

# Add soem communication
plt.title('Heart Disease Frequency per Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.ylabel('Amount')
plt.legend(['No Disease', 'Disease'])
plt.xticks(rotation = 0);
```

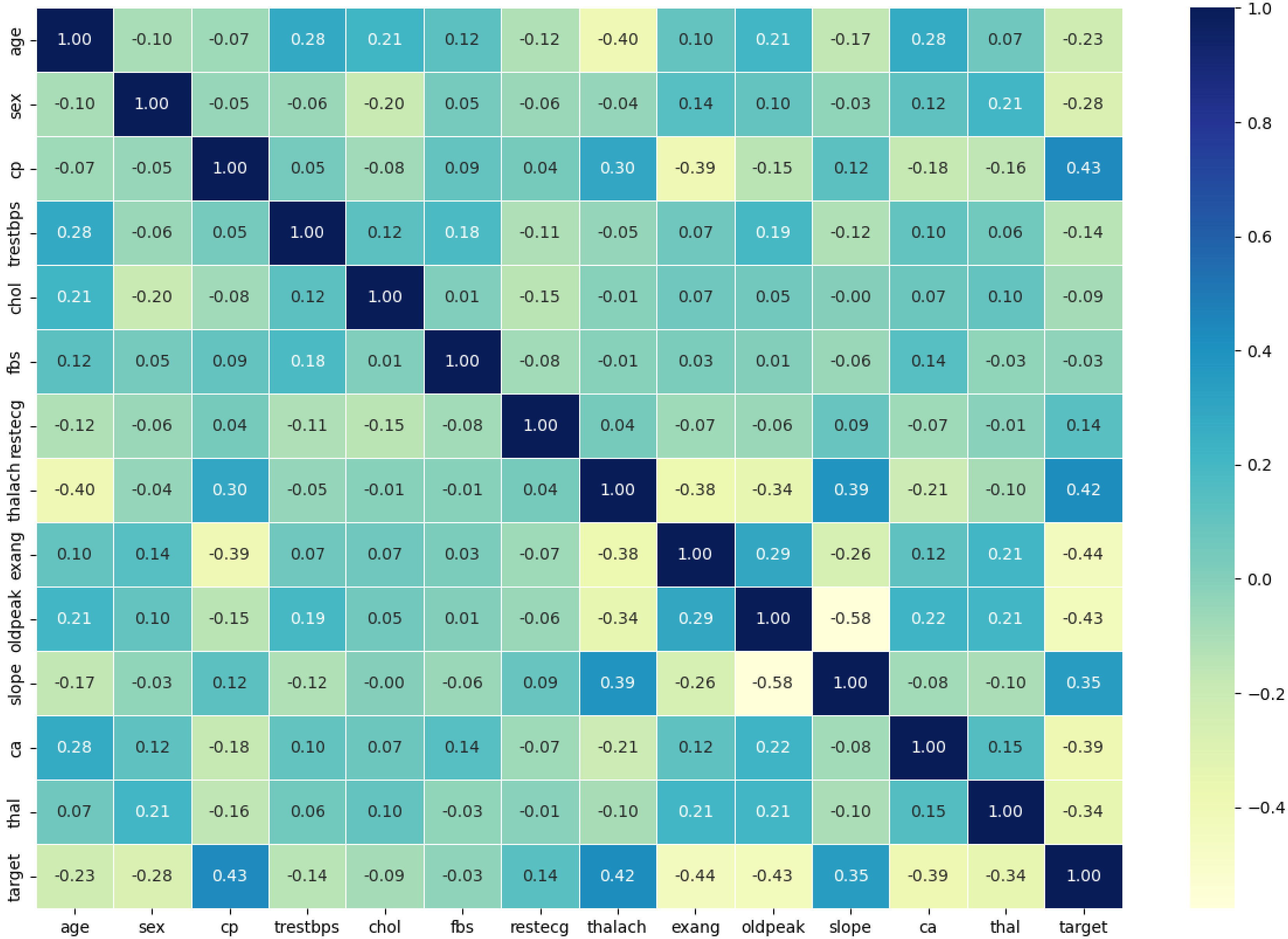

Heart Disease Frequency per Chest Pain Type
No Disease
Disease
Amount
100
80
60
40
20
0
0
1
2
3
Chest Pain Type

In [21]:
```python
# Make a correlation matrix
df.corr()
```

Out[21]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.27
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.11
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.18
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.10
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.07
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.13
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.07
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.21
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.11
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.22
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.08
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.00
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.15
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.39

In [22]:
```
# let's make our correlation matrix a little prettier

corr_matrix = df.corr()
fig, ax = plt.subplots(figsize = (15, 10))
ax = sns.heatmap(corr_matrix,
                annot = True,
                linewidths = 0.5,
                fmt = '.2f',
                cmap = 'YlGnBu');
```


age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

5. Modelling

In [23]:
```
df.head()
```

Out[23]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [24]:
```
# split data into x and y

x = df.drop('target', axis = 1)

y = df['target']
```

In [25]:
```
x.head()
```

Out[25]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

In [26]:
```
y
```

Out[26]:
```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

In [27]:
```
# split data into train and test sets
np.random.seed(42)

# split into train and test set
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.2)
```

In [28]:
```
x_train
```

Out[28]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
132	42	1	1	120	295	0	1	162	0	0.0	2	0	2
202	58	1	0	150	270	0	0	111	1	0.8	2	0	3
196	46	1	2	150	231	0	1	147	0	3.6	1	0	2
75	55	0	1	135	250	0	0	161	0	1.4	1	0	2
176	60	1	0	117	230	1	1	160	1	1.4	2	2	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
188	50	1	2	140	233	0	1	163	0	0.6	1	1	3
71	51	1	2	94	227	0	1	154	1	0.0	2	1	3
106	69	1	3	160	234	1	0	131	0	0.1	1	1	2
270	46	1	0	120	249	0	0	144	0	0.8	2	0	3
102	63	0	1	140	195	0	1	179	0	0.0	2	2	2

242 rows × 13 columns

In [29]:
```
y_train, len(y_train)
```

Out[29]:
```
(132    1
 202    0
 196    0
 75     1
 176    0
       ..
 188    0
 71     1
 106    1
 270    0
 102    1
 Name: target, Length: 242, dtype: int64,
 242)
```

Now we've got our data split into train and test sets, it's time to build a machine learning model.

We'll train it (find the patterns) on the training set.

And we'll test it (use the patterns) on the test set.

we're going to try 3 different machine learning models:

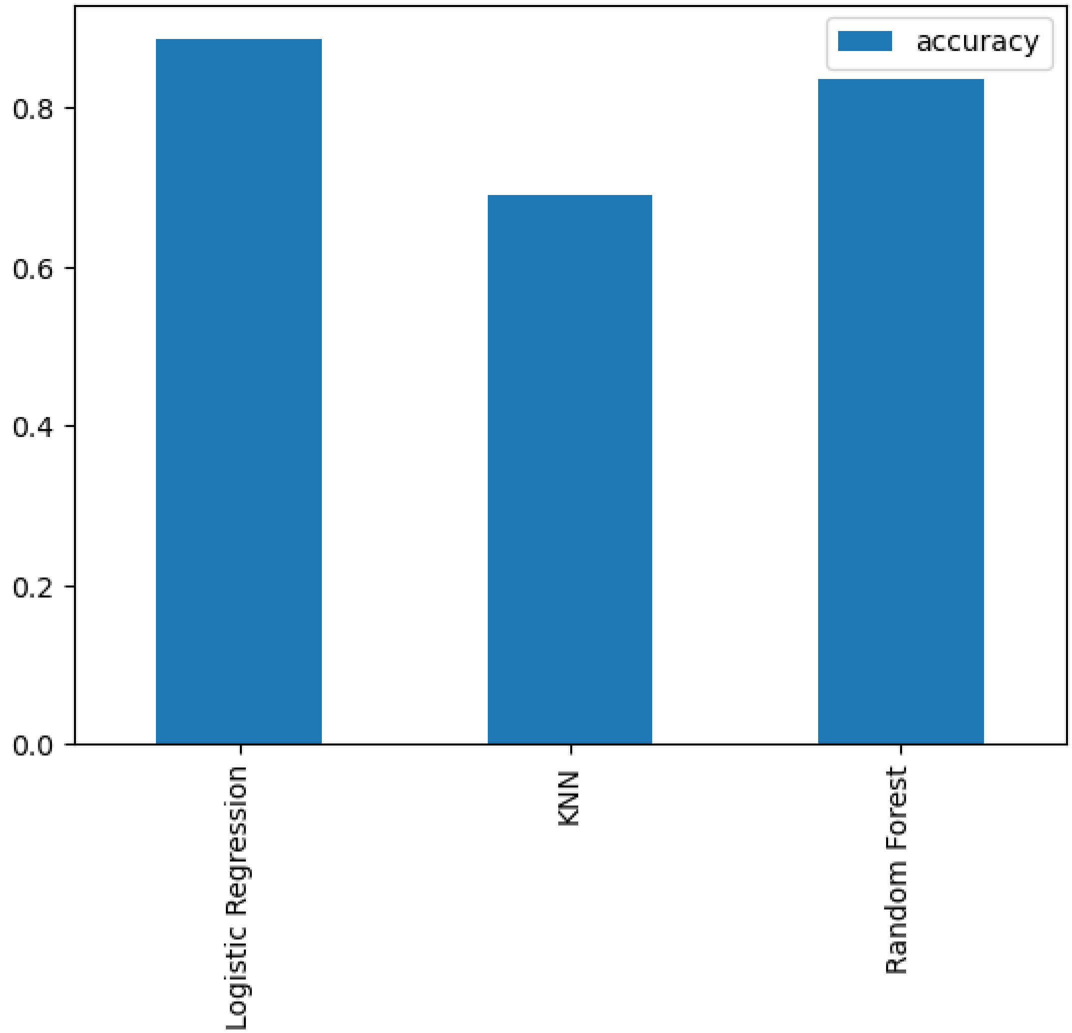
1. Logistic Regression
2. k-Nearest Neighbours Classifier
3. Random Forest Classifier

In [30]:
```
# Put models in a dictionary

models = {'Logistic Regression': LogisticRegression(),
          'KNN': KNeighborsClassifier(),
          'Random Forest': RandomForestClassifier()}

# Create a function to fit and score models
def fit_and_score(models, x_train, x_test, y_train, y_test):
    '''
    Fits and evaluate given machine learning models.
    models: a dict of different scikit-learn machine learning models
    x_train: training data (no labels)
    x_test: testing data (no labels)
    y_train: training labels
    y_test: testing labels
    '''
    # set random seed
    np.random.seed(42)
    # make a dictionary to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # fit the model to the data
        model.fit(x_train, y_train)
        # Evaluate the model and append its score to model_score
        model_scores[name] = model.score(x_test, y_test)
    return model_scores
```

In [82]:
```
model_scores = fit_and_score(models = models,
                             x_train = x_train,
                             x_test = x_test,
                             y_train = y_train,
                             y_test = y_test)
model_scores
warnings.filterwarnings('ignore')
```

Model Comparison

In [32]:
```
model_compare = pd.DataFrame(model_scores, index=['accuracy'])
model_compare.T.plot.bar();
```



Now we've got a baseline model.. and we know a model's first predictions aren't always what we should based our next steps off. What should do?

let's look at the following:

- Hyperparameter tuning

- features importance
- Confusion matrix
- cross-validation
- precision
- recall
- f1 score
- classification
- ROC Curve
- Area under the curve (AUC)

Hyperparameter tuning

In [64]:

```python
# Let's tune KNN

train_scores = []
test_scores = []

# create a list of different values for n_neighbors
neighbors = range(1, 21)

# setup KNN instance
knn = KNeighborsClassifier()

# Loop through different n_neighbors
for i in neighbors:
    knn.set_params(n_neighbors=i)

    # fit the algorithm
    knn.fit(x_train, y_train)

    # update the training scores list
    train_scores.append(knn.score(x_train, y_train))

    # update the test scores list
    test_scores.append(knn.score(x_test, y_test))
    import warnings
    warnings.filterwarnings('ignore')
```

In [34]:
```
train_scores
```

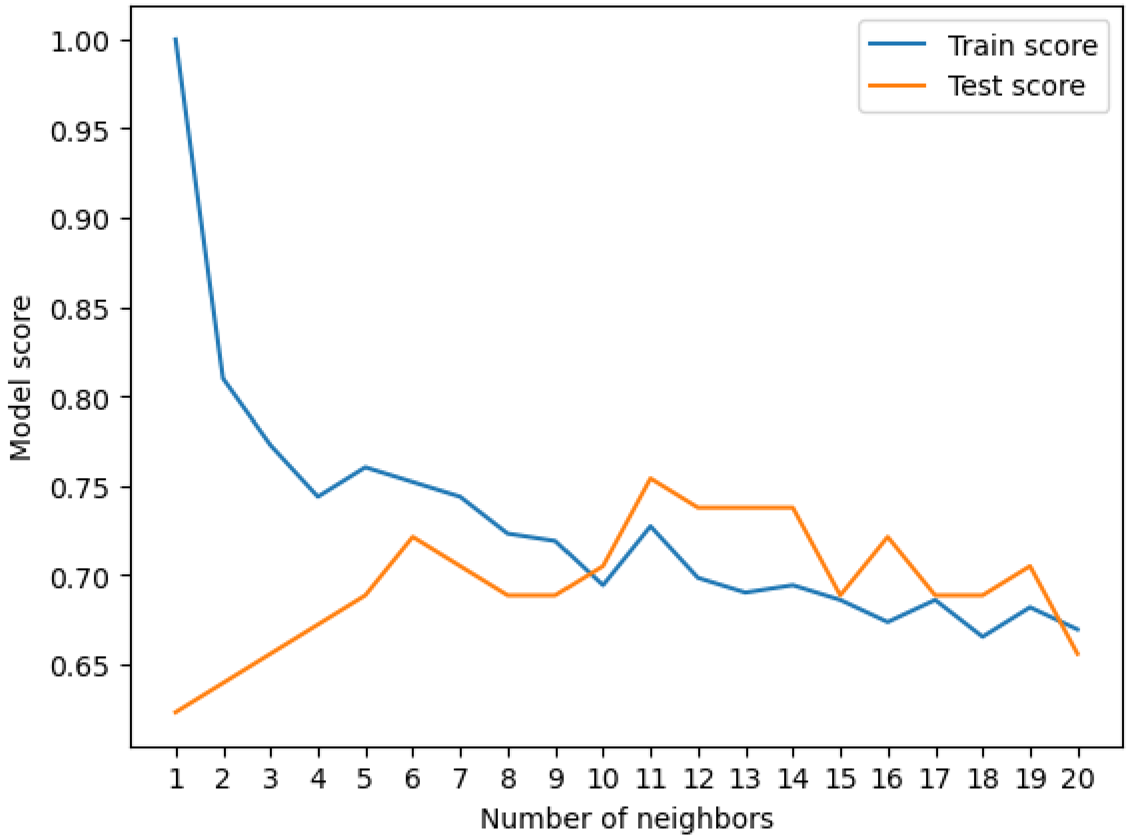
Out[34]:
```
[1.0,
 0.8099173553719008,
 0.7727272727272727,
 0.743801652892562,
 0.7603305785123967,
 0.7520661157024794,
 0.743801652892562,
 0.7231404958677686,
 0.71900826446281,
 0.6942148760330579,
 0.7272727272727273,
 0.6983471074380165,
 0.6900826446280992,
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

In [35]:
```
test_scores
```

Out[35]:
```
[0.6229508196721312,
 0.639344262295082,
 0.6557377049180327,
 0.6721311475409836,
 0.6885245901639344,
 0.7213114754098361,
 0.7049180327868853,
 0.6885245901639344,
 0.6885245901639344,
 0.7049180327868853,
 0.7540983606557377,
 0.7377049180327869,
 0.7377049180327869,
 0.7377049180327869,
 0.6885245901639344,
 0.7213114754098361,
 0.6885245901639344,
 0.6885245901639344,
 0.7049180327868853,
 0.6557377049180327]
```

In [36]:
```
plt.plot(neighbors, train_scores, label = 'Train score')
plt.plot(neighbors, test_scores, label = 'Test score')
plt.xticks(np.arange(1, 21, 1))
plt.xlabel('Number of neighbors')
plt.ylabel('Model score')
plt.legend()

print(f'Maximum KNN score on the test data: {max(test_scores) * 100:.2f}%')
```

```
Maximum KNN score on the test data: 75.41%
```



Hyperparameter tuning with RandomizedsearchCV

we're going to tune:

- LogisticRegression
- RandomForestClassifier

... using RandomizedSearchCV

In [37]:
```
# Create a hyperparameter grid for LogisticRegression
log_reg_grid = {'C': np.logspace(-4, 4, 20),
               'solver': ['liblinear']}

# create a hyperparameter grid for RandomForestClasssifier
rf_grid = {'n_estimators': np.arange(10, 1000, 50),
          'max_depth': [None, 3, 5, 10],
          'min_samples_split': np.arange(2, 20, 2),
          'min_samples_leaf': np.arange(1, 20, 2)}
```

Now we've got hyperparameter grids setup for each of our models, RandomizedSearchCV...

In [38]:
```
# Tune LogisticRegression
np.random.seed(42)

# setup random hyperparameter search for logisticregression
rs_log_reg = RandomizedSearchCV(LogisticRegression(),
                               param_distributions = log_reg_grid,
                               cv=5,
                               n_iter=20,
                               verbose=True)

# Fit random hyperparameter search model for LogisticRegression
rs_log_reg.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[38]:
```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                   param_distributions={'C': array([1.00000000e-04, 2.63665090e-04, 6.95192796e-04, 1.83
298071e-03,
       4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
       2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
       1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
       5.45559478e+02, 1.43844989e+03, 3.79269019e+03, 1.00000000e+04]),
                                        'solver': ['liblinear']},
                   verbose=True)
```

In [39]:
```
rs_log_reg.best_params_
```

Out[39]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [40]:
```
rs_log_reg.score(x_test, y_test)
```

Out[40]:
```
0.8852459016393442
```

Now, we've tuned Logisticregression(), let's do the same for RandomforestClassifier

In [41]:
```
# Setup random seed
np.random.seed(42)

# setup random hyperparameter search for RandomForestClassifier
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                          param_distributions = rf_grid,
                          cv=5,
                          n_iter=20,
                          verbose=True)

# fit random hyperparameter search model for RandomForestClassifier()
rs_rf.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[41]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=20,
                   param_distributions={'max_depth': [None, 3, 5, 10],
                                        'min_samples_leaf': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 1
9]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 1
8]),
                                        'n_estimators': array([ 10,  60, 110, 160, 210, 260, 310, 360, 4
10, 460, 510, 560, 610,
       660, 710, 760, 810, 860, 910, 960])},
                   verbose=True)
```

In [42]:
```
# find the best hyperparameters
rs_rf.best_params_
```

Out[42]:
```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

In [43]:
```
# Evaluation the randomized search RandomForestClassifier model
rs_rf.score(x_test, y_test)
```

Out[43]:
```
0.8688524590163934
```

In [44]:
```
model_scores
```

Out[44]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

Hyperparameters using GridSearchCV

since our LogisticRegression model provides the best scores so far, we'll try and improve them again using GridSearchCV...

In [45]:
```
# Different hyperparamters for our LogisticRegression
log_reg_grid = {'C': np.logspace(-4, 4, 30),
                'solver': ['liblinear']}

# setup grid hyperparameter search for LogisticRegression
gs_log_reg = GridSearchCV(LogisticRegression(),
                          param_grid = log_reg_grid,
                          cv=5,
                          verbose=True)

# fit grid hyperparameter search model
gs_log_reg.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
```

Out[45]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.00000000e-04, 1.88739182e-04, 3.56224789e-04, 6.72335754e-04,
       1.26896100e-03, 2.39502662e-03, 4.52035366e-03, 8.53167852e-03,
       1.61026203e-02, 3.03919538e-02, 5.73615251e-02, 1.08263673e-01,
       2.04335972e-01, 3.85662042e-01, 7.27895384e-01, 1.37382380e+00,
       2.59294380e+00, 4.89390092e+00, 9.23670857e+00, 1.74332882e+01,
       3.29034456e+01, 6.21016942e+01, 1.17210230e+02, 2.21221629e+02,
       4.17531894e+02, 7.88046282e+02, 1.48735211e+03, 2.80721620e+03,
       5.29831691e+03, 1.00000000e+04]),
                         'solver': ['liblinear']},
             verbose=True)
```

In [46]:
```
# check the best hyperparameters
gs_log_reg.best_params_
```

Out[46]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [47]:
```
# Evaluate the grid search Logisticregression model
gs_log_reg.score(x_test, y_test)
```

Out[47]:
```
0.8852459016393442
```

Evaluating our tuned machine learning classifier, beyond accuracy

- ROC curve and AUC score
- Confusion matrix
- Classification report
- Precision
- Recall
- F1-score

...and it would be great if cross-validation was used where possible.

To make comparison and evaluate our model, first we need to make predictions

In [48]:

```
# make predictions with tuned model
y_preds = gs_log_reg.predict(x_test)
```

In [49]:

```
y_preds
```

Out[49]:

```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

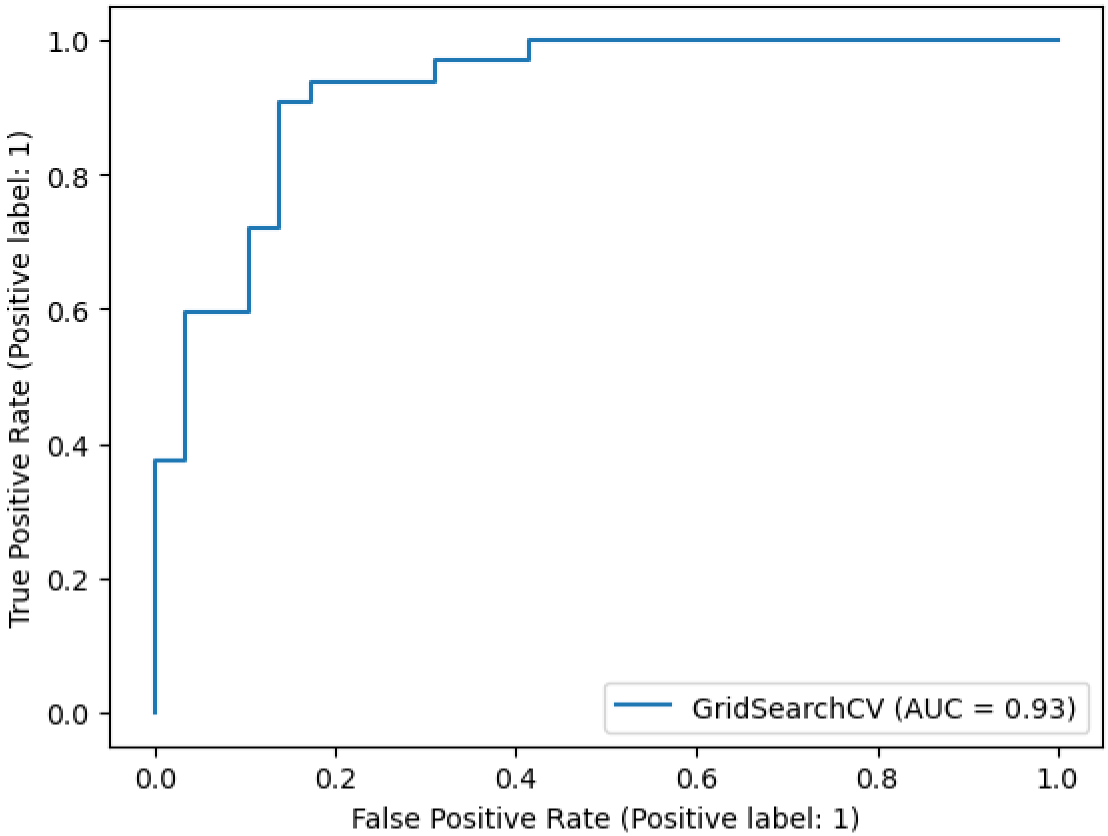
In [50]:

```
y_test
```

Out[50]:

```
179    0
228    0
111    1
246    0
60     1
      ..
249    0
104    1
300    0
193    0
184    0
Name: target, Length: 61, dtype: int64
```

In [51]:
```
# plot ROC curve and calculate and calculate AUC metrics
plot_roc_curve(gs_log_reg, x_test, y_test);
```

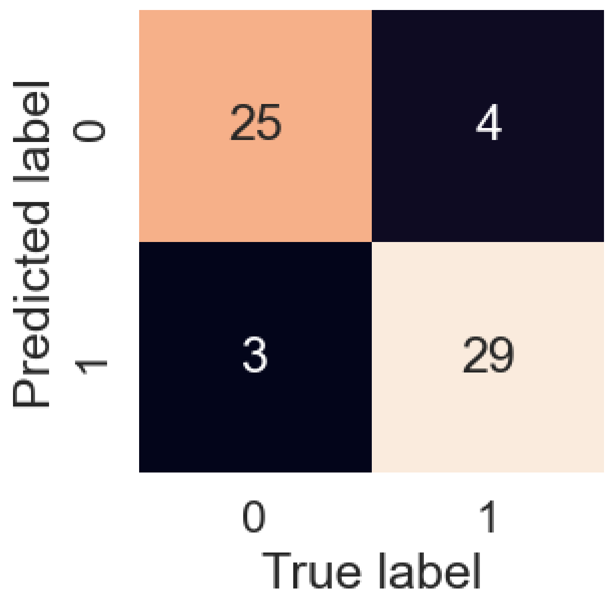


In [52]:
```
# confusion matrix
print(confusion_matrix(y_test, y_preds))
```

```
[[25  4]
 [ 3 29]]
```

In [53]:

```
sns.set(font_scale = 1.5)

def plot_conf_mat(y_test, y_preds):
    '''
    Plots a nice looking confusion matrix using seaborns's heatmap()
    '''
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, y_preds),
                     annot=True,
                     cbar=False)
    plt.xlabel('True label')
    plt.ylabel('Predicted label')
plot_conf_mat(y_test, y_preds)
```



Now we've got a ROC curve, an AUC metric and a confusion matrix, let's get classification report as well as cross-validation precision, recall and f1-score.

In [54]:
```
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

calculate evaluation metrics using cross-validation

we're going to calculate accuracy, precision, recall and f1-score of our model using cross-validation and to do so we'll be using 'cross_val_score().'

In [55]:
```
# check best hyperparameters
gs_log_reg.best_params_
```

Out[55]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [56]:
```
# create a new classifier with best parameters

clf = LogisticRegression(C=0.20433597178569418,
                         solver= 'liblinear')
```

In [57]:
```
# cross-validated accuracy

cv_acc = cross_val_score(clf,
                         x,
                         y,
                         cv=5,
                         scoring='accuracy')
cv_acc1 = (np.mean(cv_acc))
cv_acc1
```

Out[57]:
```
0.8446994535519124
```

In [58]:
```
# cross-validated precision
cv_precision = cross_val_score(clf,
                              x,
                              y,
                              cv=5,
                              scoring='precision')
cv_precision1 = np.mean(cv_precision)
cv_precision1
```

Out[58]: 0.8207936507936507

In [59]:
```
# cross-validated recall
cv_recall = cross_val_score(clf,
                           x,
                           y,
                           cv=5,
                           scoring='recall')
cv_recall1 = np.mean(cv_recall)
cv_recall1
```

Out[59]: 0.9212121212121213

In [60]:
```
# cross-validated f1-score

cv_f1_score = cross_val_score(clf,
                           x,
                           y,
                           cv=5,
                           scoring='f1')
cv_f1_score1 = np.mean(cv_recall)
cv_f1_score1
```

Out[60]: 0.9212121212121213

In [61]:

```
# vitualize cross-validated-metrics
cv_metrics = pd.DataFrame({'Accuracy': cv_acc1,
                           'Precision': cv_precision1,
                           'Recall': cv_recall1,
                           'F1': cv_f1_score1},
                         index=[0] )
cv_metrics
```

Out[61]:

	Accuracy	Precision	Recall	F1
0	0.844699	0.820794	0.921212	0.921212

In [62]:

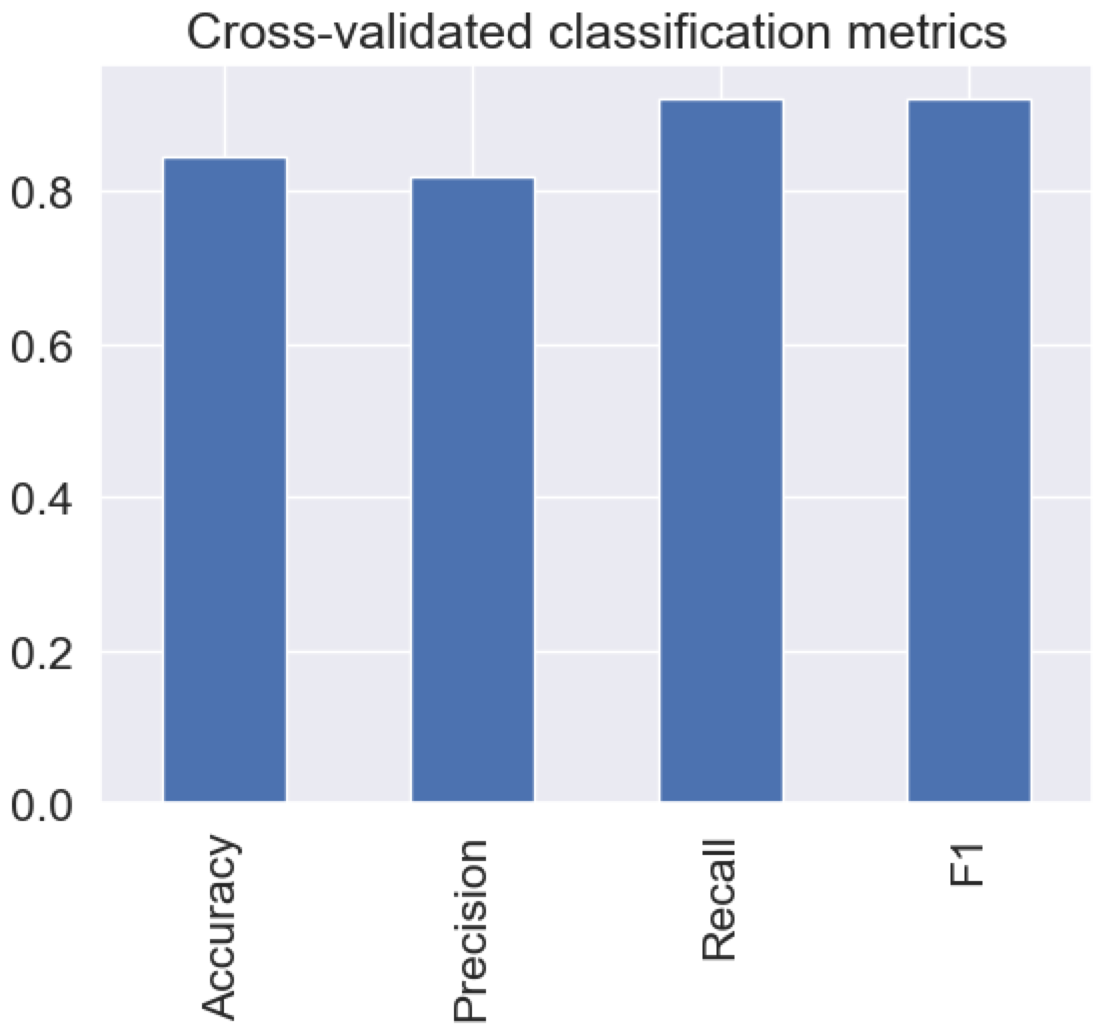
```
# vitualize cross-validated-metrics
cv_metrics = pd.DataFrame({'Accuracy': cv_acc,
                           'Precision': cv_precision,
                           'Recall': cv_recall,
                           'F1': cv_f1_score},
                        )
cv_metrics
```

Out[62]:

	Accuracy	Precision	Recall	F1
0	0.819672	0.775000	0.939394	0.849315
1	0.901639	0.885714	0.939394	0.911765
2	0.868852	0.857143	0.909091	0.882353
3	0.883333	0.861111	0.939394	0.898551
4	0.750000	0.725000	0.878788	0.794521

In [63]:

```
# vitualize cross-validated-metrics
cv_metrics = pd.DataFrame({'Accuracy': cv_acc1,
                          'Precision': cv_precision1,
                          'Recall': cv_recall1,
                          'F1': cv_f1_score1},
                        index=[0] )
cv_metrics.T.plot.bar(title='Cross-validated classification metrics',
                     legend = False);
```



Feature importance

feature importance is another way of asking, 'which features contributed most to the outcome of the model and how did they contribute?'

Finding feature importance is different for each machine learning models. One way to find feature importance is to search for '(Model NAME)' feature importance.

let's find the feature importance for our LogisticRegression model...which gave the best result out of the three models

In [65]:
```
df.head()
```

Out[65]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [66]:
```
# Fit an instance of LogisticRegression
gs_log_reg.best_params_


```

Out[66]: {'C': 0.20433597178569418, 'solver': 'liblinear'}

In [67]:
```
clf = LogisticRegression(C=0.20433597178569418,
                         solver='liblinear')
clf.fit(x_train, y_train);
```

In [68]:
```
# check coef_
clf.coef_
```

Out[68]:
```
array([[ 0.00316728, -0.86044655,  0.66067042, -0.01156993, -0.00166374,
         0.04386109,  0.31275847,  0.02459361, -0.60413083, -0.56862804,
         0.4505163 , -0.63609898, -0.67663378]])
```

In [74]:
```
df.head()
```

Out[74]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [73]:
```
# match coef's of features to columns
feature_dict = dict(zip(df.columns, list(clf.coef_[0])))
feature_dict
```

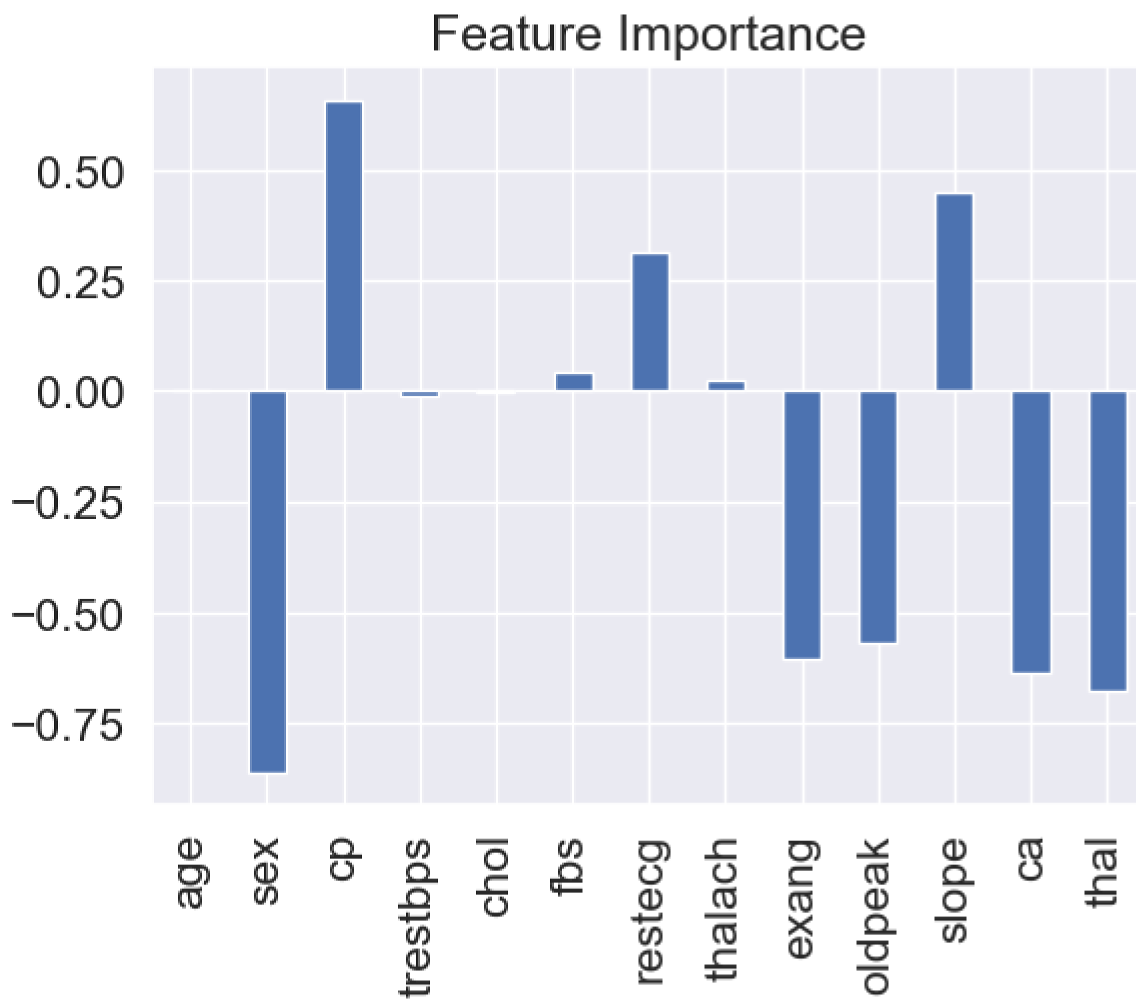
Out[73]:
```
{'age': 0.0031672809701328098,
 'sex': -0.8604465542018854,
 'cp': 0.6606704161071124,
 'trestbps': -0.011569931839584581,
 'chol': -0.0016637442846940298,
 'fbs': 0.043861090099753856,
 'restecg': 0.3127584688139112,
 'thalach': 0.024593614036076978,
 'exang': -0.6041308274033194,
 'oldpeak': -0.5686280446250761,
 'slope': 0.45051629703183155,
 'ca': -0.6360989766185763,
 'thal': -0.6766337834775279}
```

In [75]:
```
# visualize featurte importance

feature_df = pd.DataFrame(feature_dict, index=[0])
feature_df
```

Out[75]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	t
0	0.003167	-0.860447	0.66067	-0.01157	-0.001664	0.043861	0.312758	0.024594	-0.604131	-0.568628	0.450516	-0.636099	-0.676

In [77]:
```
feature_df.T.plot.bar(title = 'Feature Importance', legend = False);
```



In [78]:
```
pd.crosstab(df['sex'], df['target'])
```

Out[78]:

target	0	1
sex		
0	24	72
1	114	93

In [79]:
```
pd.crosstab(df['slope'], df['target'])
```

Out[79]:

target	0	1
slope		
0	12	9
1	91	49
2	35	107

slope - the slope of the peak exercise ST segment

- 0: Upsloping: better heart rate with excercise (uncommon)
- 1: Flatsloping: minimal change (typical healthy heart)
- 2: Downslopins: signs of unhealthy heart

6. Experimentation

if you haven't hit your evaluation metric yet... ask yourself...

- could you collect more data?
- could you try a better model? like CatBoost or XGBoost?
- could you improve the current models? (beyond we've done so far)
- if your model is good enough (you have hit your evaluation metrics) how would you export it and share it with others?

In [ ]:
```
```